Optimización Multiobjetivo en Maude

Francisco Durán[1], Eva Millán[2], Javier Montes[1], and Steffen Zschaler[3]

[1] ITIS Software, Universidad de Málaga, Málaga, Spain fdm@uma.es
[2] LCC, Universidad de Málaga, Málaga, Spain emillan@uma.es
[3] ITIS Software, Universidad de Málaga, Málaga, Spain javimp2003@gmail.com
[4] King's College, London, UK steffen.zschaler@kcl.ac.uk

Resumen Este trabajo presenta un marco para la formulación y resolución de problemas de optimización multiobjetivo con restricciones utilizando Maude. Las soluciones se representan de forma nativa como términos en Maude y se transforman mediante reglas de reescritura (operadores de mutación) que, junto con una función de fitness penalizadora, permiten abordar problemas complejos. Con este planteamiento general, utilizamos algoritmos evolutivos multiobjetivo, en concreto NSGA-II y MOEA/D. Se ilustran aplicaciones en el problema de la mochila 0-1, en la asignación de responsabilidades en clases (CRA) y en la optimización de workflows.

Keywords: Optimización multiobjetivo · NSGA-II · MOEA/D · Reescritura · Maude

1. Introducción

En sistemas de reescritura como Maude se dispone de un motor para transformar modelos representados como términos utilizando reglas de reescritura. Esta capacidad no sólo permite reescribir estados, sino también comprobar la alcanzabilidad de estados que satisfacen ciertas propiedades o verificar que una propiedad se cumple en todos los estados alcanzables a partir de un estado inicial. Inspirados en ideas de Search-Based Software Engineering (SBSE) [12] y en la optimización multiobjetivo basada en heurísticos, se puede formular un problema de optimización directamente sobre términos mediante la definición de una función de fitness, como ha sido demostrado, p.ej., por Strüber et al. [16] y Burdusel et al. [2,1].

En los algoritmos evolutivos tradicionales, es común codificar las soluciones mediante vectores y definir operadores de mutación y crossover para explorar el espacio de búsqueda. Sin embargo, como se ha demostrado en herramientas como MDEOptimiser (para modelos Ecore y reglas definidas en Henshin [17]) [2], es posible utilizar el modelo directamente, es decir, representar cada individuo como un modelo que va evolucionando. En este caso, las mutaciones se implementan como operaciones de refactorización definidas mediante reglas de reescritura. Al utilizar los estados directamente como los individuos del algoritmo, aunque para problemas concretos quizás pudiera definirse de forma ad hoc,

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

es complicado definir operadores de crossover sobre ellos, aunque puede haber problemas en los que sea posible definirlos. La gran ventaja de este enfoque es que cuando se trata de modelos complejos, las representaciones como vectores pueden ser complicadas, y las operaciones de mutación genéricas llevan con demasiada facilidad a modelos no válidos, lo que hace que sea necesario descartar una gran cantidad de los individuos generados, obligando a realizar acciones de reparación de los modelos, lo cual reduce sustancialmente la eficiencia de la búsqueda. Por el contrario, al utilizar directamente los modelos, las operaciones de mutación pueden ser más sofisticadas y especializadas, incluso garantizar que todos los estados generados son válidos, entendiendo por validez cualquier criterio que se quiera considerar. P.ej., en [13] se utiliza este enfoque, utilizando un conjunto de operadores de mutación que garantizan la consistencia de las configuraciones de productos en líneas de productos software.

El presente trabajo presenta un enfoque unificado en el que se utiliza Maude para definir problemas de optimización multiobjetivo con restricciones, aprovechando tanto sus capacidades de reescritura como de verificación formal. Se discuten las principales estrategias de búsqueda –desde métodos locales como hill climbing y simulated annealing hasta algoritmos evolutivos multiobjetivo [5]– y se analizan las implicaciones de prescindir del crossover en favor de mutaciones robustas y una función de fitness bien definida. La implementación del sistema se ha realizado instanciando el marco de programación multiobjetivo jMetal. Se ilustra el enfoque con tres problemas de optimización multiobjetivo con restricciones: el problema de la mochila, la asignación de responsabilidades en clases (CRA) y la optimización de workflows. Se muestran resultados experimentales para estos tres problemas utilizando NSGA-II [6] y MOEA/D [18], pero jMetal [10] ofrece una amplia variedad de algoritmos de optimización multiobjetivo (evolutivos y con enjambre de partículas) y monobjetivo que igualmente podrían ser utilizados.

2. Preliminares sobre optimización multiobjetivo y Maude

2.1. Formulación del problema multiobjetivo con restricciones

Un problema de optimización multiobjetivo se define generalmente como:

$$\begin{array}{rl} \text{Optimizar} & f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \\ \text{sujeto a} & g_i(x) \leq 0, \quad i = 1, \ldots, p, \\ & h_j(x) = 0, \quad j = 1, \ldots, q, \\ & x \in X. \end{array}$$

donde f es la función a optimizar y los f_i, con $i = 1..m$ son los m objetivos a satisfacer (típicamente se habla de minimizar, si se desea maximizar algún componente se usa como $-f_i(x)$). Aunque normalmente x es un vector, en nuestro caso, x representa una solución modelada como un término en Maude. Las

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

restricciones se integran en la evaluación mediante penalizaciones en la función de fitness, de modo que las soluciones que violan alguna restricción obtengan valores desfavorables.

2.2. Lógica de reescritura y Maude

Maude [4,3,7] es un lenguaje de especificación basado en la lógica de reescritura [14], una lógica de cambio que puede tratar con estados y cálculos concurrentes no deterministas de una forma muy natural. Una teoría lógica de reescritura es una tupla $(\Sigma;E;R)$, donde $(\Sigma;E)$ es una teoría ecuacional que especifica los estados del sistema como elementos del álgebra inicial $T_{(\Sigma;\mathrm{E})}$, y R es un conjunto de reglas de reescritura que describen las posibles transiciones simultáneas de un solo paso en el sistema. Σ es una signatura que especifica la estructura de tipos (tipos y subtipos) y operaciones, y E es la colección de ecuaciones y axiomas de pertenencia declaradas en el módulo funcional. Las especificaciones de reescritura así descritas son ejecutables si satisfacen restricciones tales como la terminación y confluencia de la subespecificación ecuacional, y la coherencia de ecuaciones y reglas. Maude proporciona soporte para la reescritura módulo asociatividad, conmutatividad e identidad.

Dada una especificación de reescritura, Maude puede ser utilizada para realizar una variedad de tareas, como simulación (reescribiendo términos) y verificación formal (podemos verificar propiedades como la confluencia y la terminación de las reglas de reescritura, y también realizar análisis de alcanzabilidad, model checking y demostración de teoremas). Con la presente propuesta, además, es posible utilizar Maude para resolver problemas de optimización multiobjetivo con restricciones, aprovechando la capacidad de reescritura para generar nuevas soluciones.

En Maude, los sistemas orientados a objetos pueden especificarse mediante módulos orientados a objetos en los que se declaran clases y subclases, con el soporte habitual para la herencia y la vinculación dinámica. Una clase se declara con la sintaxis $\mathsf{class}\ C \mid a_1\colon S_1, \ldots, a_n\colon S_n$, donde C es el nombre de la clase, a_i son los identificadores de los atributos y S_i son los tipos de los atributos correspondientes. Los objetos de una clase C son estructuras similares a registros de la forma $< O : C \mid a_1\colon v_1, \ldots, a_n\colon v_n >$, donde O es el identificador del objeto, y v_i son los valores actuales de sus atributos.

En Maude, los sistemas orientados a objetos se axiomatizan mediante teorías ecuacionales que describen sus estados como tipos de datos algebraicos y colecciones de reglas de reescritura condicionales que especifican su *comportamiento*. Las reglas de reescritura se escriben como

$$\mathsf{crl}\ [l] : T \Rightarrow T'\ \mathsf{if}\ C$$

donde l es la etiqueta de la regla, T y T' son términos, y C es su guardia o condición.

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

2.3. Representación de soluciones en Maude

La utilización de Maude permite que las soluciones se representen de forma nativa como términos, lo que integra la modelización y la verificación formal en un mismo marco. Para ilustrar el enfoque, consideremos el problema de la mochila 0/1. El problema consiste en una mochila con capacidad máxima W y un conjunto de n objetos, cada uno con un valor v_i y un peso w_i. Como problema multiobjetivo, el problema se puede formular como la maximización del valor total y la minimización del número de objetos en la mochila, sujeto a que el peso total no exceda la capacidad de la mochila. Podemos representar el problema en Maude como un objeto de la siguiente clase `Knapsack`:

```
class Knapsack | numItems : Nat, capacity : Float,
         weights : Map{Nat, Float}, profits : Map{Nat, Float} .
```

Podemos representar una solución como un término de la clase `Solution`:

```
class Solution | bits : Map{Nat, Bool} .
```

En el caso de la mochila, los posibles cambios en la solución se pueden modelar mediante reglas de reescritura que añaden o eliminan objetos de la mochila, es decir, que cambian el valor de los bits que indican si un objeto está en la mochila o no.

```
rl [mutate] :
   < O : Solution | bits : (I |-> Bit, Bits) >
=> < O : Solution | bits : (I |-> (not Bit), Bits) > .
```

Los objetivos del problema consisten en maximizar el valor total de los objetos en la mochila y minimizar el número de objetos. La función de fitness se puede definir como:

$$f(\text{solución}) = (-\text{valor total}, \text{número de objetos}),$$

Proporcionamos sendas funciones que realicen dichos cálculos (supuestas funciones auxiliares `totalWeight`, `totalProfit` y `numOfItems` que calculan, respectivamente, el peso total de los objetos en la mochila, el valor total y el número de objetos en la mochila):

```
op evalObje0 : JMetalTerm -> Float .
eq evalObje0(< O1 : Knapsack | capacity : Capacity > < O2 : Solution | >)
 = if totalWeight(< O1 : Knapsack | > < O2 : Solution | >) > Capacity
   then 0.0
   else - totalProfit(< O1 : Knapsack | > < O2 : Solution | >)
   fi .

op evalObje1 : JMetalTerm -> Float .
eq evalObje1(< O1 : Knapsack | capacity : Capacity > < O2 : Solution | >)
 = numOfItems(< O1 : Knapsack | > < O2 : Solution | >) .
```

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

Por último, el problema queda sujeto a las restricciones establecidas. En este caso, no superar la capacidad de la mochila:

```
op evalCons0 : JMetalTerm -> Float .
eq evalCons0(
     < O1 : Knapsack | capacity : Capacity >
     < O2 : Solution | >)
  = Capacity - totalWeight(< O1 : Knapsack | > < O2 : Solution | >) .
```

2.4. El marco de trabajo JMetal

JMetal [10] es un marco de trabajo para la resolución de problemas de optimización multiobjetivo y monobjetivo. El marco proporciona una serie de interfaces, clases abstractas y algoritmos de optimización multiobjetivo y monobjetivo, así como una serie de operadores de mutación y crossover, que nos permiten instanciar problemas de distinta naturaleza según nuestras necesidades.

Para simplificar la comunicación de JMetal con Maude, se ha utilizado la versión Python de JMetal, jMetalPy [10], de forma que utilizando la librería `maude` de Python [15], podemos controlar Maude desde Python y realizar las operaciones necesarias para la evaluación de las soluciones y la generación de nuevas soluciones.

Para poder instanciar el marco con el problema de la mochila, definimos:

- Una clase `MaudeSolution` que extiende la clase `Solution` de jMetalPy, y que básicamente contiene un término Maude que representa la solución y que proporciona operaciones, p.ej., para copiar soluciones.
- Una clase `MaudeProblem` que extiende la clase `Problem` de jMetalPy, y que contiene la lógica necesaria para evaluar las soluciones, evaluar objetivos y constraints invocando a las funciones definidas en Maude, y generar nuevas soluciones.
- Una clase `MaudeMutation` que extiende la clase `Mutation` de jMetalPy, y que implementa la mutación de las soluciones. El método `execute` de esta clase invoca a Maude para obtener todas las posibles reescrituras de la solución que recibe como argumento y selecciona una de ellas.
- Una clase `MaudeCrossover` que extiende la clase `Crossover` de jMetalPy, y que implementa el crossover de las soluciones. En nuestro caso hemos dejado el método `execute` de esta clase sin implementar, por lo que no se realizarán operaciones de recombinación de individuos. En este caso no hubiera sido complicado hacerlo, sin embargo, no es lo habitual. Los otros dos ejemplos que presentamos en las siguientes secciones presentan situaciones bastante más complejas.

Una instancia del problema y una instancia del algoritmo deseado completan la configuración necesaria para ejecutar el algoritmo y obtener los resultados.

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

2.5. Estrategias de búsqueda y consideraciones sobre el crossover

Aunque el crossover es el mecanismo que permite compartir información de diferentes regiones del espacio de soluciones entre candidatos, dado que la complejidad de la representación en Maude dificulta la implementación de un operador de *crossover* general, nuestro enfoque se centra en estrategias de búsqueda que operan exclusivamente con mutaciones. Al eliminarlo, el algoritmo corre el riesgo de convertirse en una búsqueda local aleatorizada aplicada en paralelo a cada individuo. Para que este esquema sea efectivo, se deben incorporar estrategias de selección robustas, tales como:

- *Selección y Reemplazo:* Mecanismos como el elitismo aseguran que los mejores individuos se conserven, mientras que estrategias de nicho ayudan a mantener la diversidad.
- *Búsqueda Local:* Métodos como hill climbing y simulated annealing pueden explorar el vecindario de cada solución, aunque la ausencia de crossover puede hacer que la convergencia sea más lenta y la exploración del espacio menos amplia.
- *Algoritmos evolutivos multiobjetivo sin crossover:* Alternativas como NSGA-II, NSGA-III, AGEA y AMOEA están diseñado para hacer una selección elitista y diversificada. P.ej., AGEA y AMOEA utilizan mecanismos de nicho para preservar la diversidad, lo cual es esencial cuando la única fuente de variación es la mutación.

En resumen, aunque la eliminación del crossover reduce la capacidad de combinar información de distintos candidatos, la aplicación de mutación exclusiva, combinada con mecanismos de selección y preservación de diversidad, puede compensar esta limitación y permitir una exploración efectiva del espacio de soluciones.

Al margen de la selección, las otras dos piezas fundamentales para garantizar un comportamiento aceptable son las operaciones de mutación y la función de fitness. En cuanto a las primeras, al operar directamente sobre el modelo, es posible proporcionar operaciones sofisticadas, especializadas para el problema en concreto. En [16], Strüber propone un método para generar un conjunto óptimo de operaciones de mutación específicas para un problema determinado. En [1], Burdusel propone usar heurísticos para guiar el proceso de mutación.

3. Experimentación

Para ilustrar el marco propuesto, se consideran tres casos de aplicación: el problema de la mochila presentado en la sección anterior, el Class Responsibility Assignment y la optimización de workflows.

El Class Responsibility Assignment (CRA) [11] es un problema de optimización propuesto en la 9th Transformation Tool Contest (TTC 2016). Dado un conjunto de características (métodos y atributos), con unas dependencias establecidas entre ellas (métodos que usan variables o invocan otras funciones), se

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

trata de asignar dichas características a clases, de forma que se maximice la cohesión y se minimice el acoplamiento, respetando restricciones de diseño, como el número de atributos no-encapsulados o el número de clases vacías. En este caso, las operaciones de mutación representan operaciones de refactorización que permiten crear nuevas clases, asignar características a clases, mover características de una clase a otra, etc.

El problema de la optimización de workflows consiste en aplicar operaciones de refactorización a un workflow, de forma que se minimice el tiempo total de ejecución y el coste total, respetando restricciones en la secuencia de tareas (dependencias duras). Las operaciones de refactorización extienden las presentadas en [9]. Estas operaciones de refactorización reorganizan flujos y tareas, por ejemplo, poniendo en paralelo dos tareas que estaban en secuencia, moviendo o sacando una tarea de una estructura paralela, etc. siempre que se satisfagan las dependencias duras establecidas. Los workflows incluyen información sobre duración de tareas, retrasos en los flujos, costes de ejecución y necesidades y disponibilidad de recursos. Con esta información, es posible calcular el tiempo total de ejecución y el coste total de un workflow como se describe en [8,**?**].

Con los problemas de optimización propuestos como se ha descrito en las secciones anteriores, podemos aplicar algoritmos de optimización multiobjetivo utilizando el marco de trabajo jMetalPy. En concreto, se han aplicado los algoritmos NSGA-II y MOEA/D, que son dos de los algoritmos más utilizados en la literatura para resolver problemas de optimización multiobjetivo. Como referencia, se ha incluido un algoritmo genético con selección aleatoria (GA/R). Las tablas 1-3 muestran los resultados obtenidos para los experimentos ejecutados. En concreto, se han ejecutado 3 instancias de cada problema (mochila con 5, 50 y 500 objetos creados aleatoriamente, CRA con 3 de las instancias utilizadas en la TTC 2016, con 35, 80 y 160 características, y workflows de 10, 24 y 43 nodos). Cada uno de los algoritmos ha sido ejecutado con un máximo de 1000 evaluaciones, con poblaciones de 100 individuos y 30 descendientes en cada evaluación. Como ya hemos indicado, el problema de la mochila tiene 2 objetivos y 1 restricción, CRA tiene 2 objetivos y 2 restricciones, y workflows tiene 2 objetivos y ninguna restricción.

Como cada uno de los tres problemas tiene 2 objetivos, los algoritmos darán una o dos soluciones dependiendo de si una solución domina a todas las demás, o es necesario dos soluciones para cubrir todo el frente de Pareto. En la tabla 1 se muestran los resultados obtenidos para el problema de la mochila, y en ella podemos observar cómo sólo en el experimento de la fila 3.3 se obtienen dos soluciones. Menos en el experimento más simple de esta tabla (filas 1.1–3), en el que los tres algoritmos encuentran la misma solución, en los otros experimentos observamos una situación similar: ninguna de las soluciones domina a las demás. Aunque MOEA/D consigue el mejor valor para el objetivo 1, NSGA-II consigue el mejor valor para el objetivo 2.

La tabla 2 muestra los resultados obtenidos para el problema CRA. En este caso, la formulación del problema incluye dos objetivos y dos restricciones. Sólo NSGA-II es incapaz de encontrar una solución que domine a las demás. Sin

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

Tabla 1: Resultados de los algoritmos aplicados al problema de la mochila

							Solución 1				**Solución 2**			
Fil.	**Probl.**	**Alg.**	**Evals**	**Pobl.**	**Desc.**	**T. (s)**	Obj1	Obj2	Cons1	Cons2	Obj1	Obj2	Cons1	Cons2
1.1	4	GA/R	1000	100	30	0.07	-15.00	3.00	2.00					
1.2	4	NSGAII	1000	100	30	0.11	-15.00	3.00	2.00					
1.3	4	MOEAD	1000	100	30	0.17	-15.00	3.00	2.00					
2.1	49	GA/R	1000	100	30	0.58	-1082.00	16.00	61.00					
2.2	49	NSGAII	1000	100	30	0.65	-1018.00	14.00	58.00					
2.3	49	MOEAD	1000	100	30	2.23	-1680.00	27.00	9.00					
3.1	499	GA/R	1000	100	30	35.88	-1172.00	17.00	2169.00					
3.2	499	NSGAII	1000	100	30	32.01	-720.00	8.00	2227.00					
3.3	499	MOEAD	1000	100	30	146.29	-2186.00	35.00	2049.00		-1922.00	35.00	2025.00	

embargo, una observación más detallada nos muestra algo distinto, todas las demás soluciones son ceros. El problema CRA tiene una particularidad, y es que el cálculo de los índices de acoplamiento y cohesión vienen divididos por el producto del número de métodos y el número de atributos de una clase, lo que hace que para todas aquellas clases que no tienen un tipo de característica, atributo a método, su índice correspondiente será cero. Esto indica, simplemente, que GA/R y MOEA/D no son capaces de salir de una situación de mínimos locales, mientras que NSGA-II sí lo hace.

Tabla 2: Resultados de los algoritmos aplicados al problema CRA

							Solución 1				**Solución 2**			
Fil.	**Probl.**	**Alg.**	**Evals**	**Pobl.**	**Desc.**	**T. (s)**	Obj1	Obj2	Cons1	Cons2	Obj1	Obj2	Cons1	Cons2
1.1	RDG_C	GA/R	1000	100	30	0.24	0.00	0.00	-36.00	0.00				
1.2	RDG_C	NSGAII	1000	100	30	2.07	-2.00	4.75	-16.00	0.00	0.00	11.17	-14.00	0.00
1.3	RDG_C	MOEAD	1000	100	30	0.39	0.00	0.00	-36.00	0.00				
2.1	RDG_D	GA/R	1000	100	30	0.34	0.00	0.00	-80.00	0.00				
2.2	RDG_D	NSGAII	1000	100	30	2.43	-2.92	2.25	-58.00	0.00	-1.20	4.24	-59.00	0.00
2.3	RDG_D	MOEAD	1000	100	30	0.77	0.00	0.00	-80.00	0.00				
3.1	RDG_E	GA/R	1000	100	30	0.86	0.00	0.00	-161.00	0.00				
3.2	RDG_E	NSGAII	1000	100	30	6.79	-2.23	0.67	-139.00	0.00	-2.17	1.50	-140.00	0.00
3.3	RDG_E	MOEAD	1000	100	30	3.18	0.00	0.00	-161.00	0.00				

La tabla 3 muestra los resultados obtenidos para el problema de refactorización de workflows. Aunque la formulación del problema no tiene restricciones, es en nuestra opinión el problema más interesante de los tres. En este caso, las operaciones de mutación son complejas operaciones que involucran sofisticadas refactorizaciones y comprobaciones sobre la satisfacción de las dependencias que difícilmente podrían ser implementadas en un sistema como MDEOptimser [1]. En este caso, de nuevo, en el problema más simple (problema 1 en filas 1.1–3) los tres algoritmos encuentran la misma solución. En los otros dos, se vuelve a presentar una situación similar a la de los problemas anteriores, ninguna de las soluciones domina a las demás, y por tanto no podemos concluir que uno de los algoritmos funcione mejor que los demás. Sin embargo, hay alguna curiosidad que podemos observar: (i) la primera solución de MOEA/D (file 2.3) es mejor

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

que la primera de MOEA/D (file 2.2), (i) la solución de GA/R para el problema 4689 (fila 3.1) es mejor que la segunda solución dada por NSGA-II (fila 3.2), y (ii) para este mismo problema, la primera solución de NSGA-II (file 3.2) es mejor que la primera de MOEA/D (file 3.3).

Tabla 3: Resultados de los algoritmos aplicados al problema de workflows

							Solución 1				**Solución 2**			
Fil.	**Probl.**	**Alg.**	**Evals**	**Pobl.**	**Desc.**	**T. (s)**	Obj1	Obj2	Cons1	Cons2	Obj1	Obj2	Cons1	Cons2
1.1	1	GA/R	1000	100	30	9.28	4.00	10.00						
1.2	1	NSGAII	1000	100	30	11.12	4.00	10.00						
1.3	1	MOEAD	1000	100	30	21.67	4.00	10.00						
2.1	1936	GA/R	1000	100	30	273.07	86.00	28.00						
2.2	1936	NSGAII	1000	100	30	236.92	146.00	22.00			107.70	26.00		
2.3	1936	MOEAD	1000	100	30	276.11	138.00	22.00			89.70	35.00		
3.1	4689	GA/R	1000	100	30	384.68	99.90	49.00						
3.2	4689	NSGAII	1000	100	30	303.06	154.90	39.00			101.20	50.00		
3.3	4689	MOEAD	1000	100	30	483.15	155.40	39.00			99.60	51.00		

4. Conclusiones y trabajo futuro

El marco presentado demuestra que es posible abordar problemas de optimización multiobjetivo con restricciones utilizando Maude, aprovechando la representación de soluciones como términos y la aplicación de reglas de reescritura para la generación de nuevas soluciones. JMetal nos proporciona una amplia variedad de algoritmos de optimización multiobjetivo y monobjetivo, con una amplia variedad de operadores de selección, criterios de terminación, etc. que nos permiten instanciar problemas de distinta naturaleza según nuestras necesidades. La ausencia de un operador de crossover implica que el algoritmo depende fundamentalmente de los operadores de mutación (lo que se asemeja a una búsqueda local aleatorizada aplicada en paralelo). Por supuesto, siempre podemos aumentar el número de evaluaciones o el número de descendientes para mejorar los resultados, pero esta no es la solución más inteligente. La implementación de mecanismos de selección robustos y la incorporación de técnicas para preservar la diversidad (como los mecanismos de nicho utilizados en AGEA y AMOEA), así como tener variantes de las funciones de fitness que favorezcan evitar óptimos locales, podrían compensar esta limitación.

El éxito del enfoque depende en gran medida del ajuste fino de la función de fitness y de la tasa y naturaleza de las mutaciones, lo que abre interesantes líneas de investigación. Entre las futuras direcciones destacamos: (i) la incorporación de técnicas de aprendizaje automático para ajustar dinámicamente los parámetros de mutación y la ponderación de los objetivos, y (ii) la profundización en el análisis de estrategias de selección y la exploración de heurísticas híbridas que puedan mejorar la convergencia sin necesidad de crossover.

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial-SinDerivadas 4.0 Internacional”.

Referencias

1. Burdusel, A.: Optimisation Methods for Model-Driven Engineering. Ph.D. thesis, King's College London (2021)
2. Burdusel, A., Zschaler, S., Strüber, D.: Mdeoptimiser: a search based model engineering tool. In: Babur, Ö., Strüber, D., Abrahão, S., Burgueño, L., Gogolla, M., Greenyer, J., Kokaly, S., Kolovos, D.S., Mayerhofer, T., Zahedi, M. (eds.) Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018. pp. 12–16. ACM (2018). https://doi.org/10.1145/3270112.3270130, https://doi.org/10.1145/3270112.3270130
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude, LNCS, vol. 4350. Springer (2007)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and programming in rewriting logic. Theoretical Computer Science **285**(2), 187–243 (2002)
5. Deb, K.: Multi-objective optimisation using evolutionary algorithms: An introduction. In: Wang, L., Ng, A.H.C., Deb, K. (eds.) Multi-objective Evolutionary Optimisation for Product Design and Manufacturing, pp. 3–34. Springer (2011). https://doi.org/10.1007/978-0-85729-652-8_1, https://doi.org/10.1007/978-0-85729-652-8_1
6. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017, https://doi.org/10.1109/4235.996017
7. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.: Programming and symbolic computation in Maude. J. Log. Algebraic Methods Program. **110** (2020). https://doi.org/10.1016/J.JLAMP.2019.100497
8. Durán, F., Salaün, G.: Verifying timed BPMN processes using maude. In: Jacquet, J., Massink, M. (eds.) Coordination Models and Languages - 19th IFIP WG 6.1 International Conference, COORDINATION 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10319, pp. 219–236. Springer (2017). https://doi.org/10.1007/978-3-319-59746-1_12, https://doi.org/10.1007/978-3-319-59746-1_12
9. Durán, F., Salaün, G.: Optimization of BPMN processes via automated refactoring. In: Troya, J., Medjahed, B., Piattini, M., Yao, L., Fernández, P., Ruiz-Cortés, A. (eds.) Service-Oriented Computing - 20th International Conference, ICSOC 2022, Seville, Spain, November 29 - December 2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13740, pp. 3–18. Springer (2022). https://doi.org/10.1007/978-3-031-20984-0_1, https://doi.org/10.1007/978-3-031-20984-0_1
10. Durillo, J.J., Nebro, A.J., Alba, E.: The jmetal framework for multi-objective optimization: Design and architecture. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010. pp. 1–8. IEEE (2010). https://doi.org/10.1109/CEC.2010.5586354, https://doi.org/10.1109/CEC.2010.5586354
11. Fleck, M., Troya, J., Wimmer, M.: The class responsibility assignment case. In: García-Domínguez, A., Krikava, F., Rose, L.M. (eds.) Proceedings of the

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".

9th Transformation Tool Contest, co-located with the 2016 Software Technologies: Applications and Foundations (STAF 2016), Vienna, Austria, July 8, 2016. CEUR Workshop Proceedings, vol. 1758, pp. 1–8. CEUR-WS.org (2016), https://ceur-ws.org/Vol-1758/paper1.pdf

12. Harman, M.: The current state and future of search based software engineering. In: Briand, L.C., Wolf, A.L. (eds.) International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA. pp. 342–357. IEEE Computer Society (2007). https://doi.org/10.1109/FOSE.2007.29, https://doi.org/10.1109/FOSE.2007.29
13. Horcas, J.M., Strüber, D., Burdusel, A., Martinez, J., Zschaler, S.: We're not gonna break it! consistency-preserving operators for efficient product line configuration. IEEE Trans. Software Eng. **49**(3), 1102–1117 (2023). https://doi.org/10.1109/TSE.2022.3171404, https://doi.org/10.1109/TSE.2022.3171404
14. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96**(1), 73–155 (1992)
15. Rubio, R.: Maude as a library: An efficient all-purpose programming interface. In: Bae, K. (ed.) Rewriting Logic and Its Applications - 14th International Workshop, WRLA@ETAPS 2022, Munich, Germany, April 2-3, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13252, pp. 274–294 (2022). https://doi.org/10.1007/978-3-031-12441-9_14
16. Strüber, D.: Generating efficient mutation operators for search-based model-driven engineering. In: Guerra, E., van den Brand, M. (eds.) Theory and Practice of Model Transformation - 10th International Conference, ICMT@STAF 2017, Marburg, Germany, July 17-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10374, pp. 121–137. Springer (2017). https://doi.org/10.1007/978-3-319-61473-1_9, https://doi.org/10.1007/978-3-319-61473-1_9
17. Strüber, D., Born, K., Gill, K.D., Groner, R., Kehrer, T., Ohrndorf, M., Tichy, M.: Henshin: A usability-focused framework for EMF model transformation development. In: de Lara, J., Plump, D. (eds.) Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10373, pp. 196–208. Springer (2017). https://doi.org/10.1007/978-3-319-61470-0_12, https://doi.org/10.1007/978-3-319-61470-0_12
18. Zhang, Q., Li, H.: MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. **11**(6), 712–731 (2007). https://doi.org/10.1109/TEVC.2007.892759, https://doi.org/10.1109/TEVC.2007.892759

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-SinDerivadas 4.0 Internacional".